

50 BASIC

PROGRAMMING AND THE BEGINNING OF BASIC
BASIC COMES TO THE HOME
COMMODORE BASIC
THE CIRCULATION OF BASIC PROGRAMS
LATE BASIC

The character graphics themselves, the way they line up in rows and then in columns, and even the speed at which they appear—these characteristics all contribute to the aesthetic of **10 PRINT**'s output. However, **10 PRINT** functions the way it does, in part, because it is written in a specific programming language with particular affordances and attributes: BASIC.

This “Beginner’s All-purpose Symbolic Instruction Code” has a fabled cultural and technical history. BASIC was developed by John Kemeny and Thomas Kurtz, two professors at Dartmouth College. In 1964 its creators freely shared a working version of the language, leading to its widespread adoption at the high school and college level. By that time, general-purpose computers had existed for about two decades. Many were still programmed in low-level machine languages, but high-level languages, abstracted from the idiosyncrasies of an individual machine, had also been in widespread use for a decade. BASIC continued the evolution of high-level languages, building on some of what FORTRAN, Algol, and other languages had accomplished: greater portability across platforms along with keywords and syntax that facilitated understanding the language and writing programs.

The language was developed for an early time-sharing environment, the Dartmouth Time-Sharing System (DTSS). This revolutionary configuration allowed multiple programmers to use a single system at the same time. A system of this sort—with many terminals connected to a mainframe or minicomputer—differs considerably from the personally owned, relatively inexpensive, single-user computers of the microcomputer era. But in the early 1960s, the DTSS also distinguished itself from earlier systems that required the batch processing of stacks of punched cards. Time-sharing allowed people to engage with and explore computation in significant new ways, with what felt like “real time” processing; BASIC was an important part of this computing revolution. Given the educational purpose of DTSS and BASIC, ease of use was paramount. Being easy to use helped BASIC’s massive popularity and success.

BASIC became even more influential as microcomputers entered people’s homes. In 1975 the MITS Altair 8800 computer, widely acclaimed as the first home computer, became available. Perhaps the most significant piece of software for this system was Altair BASIC, a version of BASIC that was the first product of a young company called Microsoft. Following its success with the Altair 8800, Microsoft wrote versions of BASIC for many

popular microcomputers. Thanks in large part to Microsoft, BASIC became the lingua franca of home computing. BASIC resided in the ROM of these computers, meaning a user could turn on the computer and immediately begin programming. From the late 1970s through the early 1980s, BASIC reigned supreme on home computers, with **10 PRINT** and thousands of other programs circulating through books, magazines, and computer club newsletters. BASIC was so canonical that some books of BASIC programs did not even bother to mention “BASIC” on their covers.

Despite or because of its ubiquity, BASIC has become a target of derision for many modern programmers. Inextricably associated with Microsoft and that bane of structured programmers, **GOTO**, the language has been said to encourage tangled, unmanageable code, to be unbearably slow, and to be suitable only for children and amateurs. Yet BASIC has not completely disappeared, and many programmers in the early twenty-first century remember BASIC fondly. The language was a popular success, worked well for small programs that let home users explore computing, and fostered creativity and innovation in several generations of computer users and programmers.

PROGRAMMING AND THE BEGINNING OF BASIC

While there is some dispute over who should rightly be called the first computer programmer, many have awarded this designation to Ada Byron, the Countess of Lovelace (1815–1852). She was raised by her mother and educated extensively in mathematics and logic specifically so that she might follow a different path from that of her father, Lord Byron. Such a stark separation in Ada’s upbringing offers a very early example of the perceived incompatibility between computation and poetics.

Ada Lovelace’s contributions as an early programmer are most evident in her translation of and notes to an Italian article by Louis Menabrea about Charles Babbage’s Analytical Engine (Menabrea 1842). In this work, completed in 1843, she envisioned the Analytical Engine as a general-purpose computer and described an algorithm that could be used to output Bernoulli numbers. Although the computer to execute Lovelace’s program was never built, her project made an important contribution to the modern idea of computing (Fuegi and Francis 2003). Lovelace’s “program” was an

algorithm described in mathematical notation.

The computer programs that followed in the electromechanical and early electronic age of computing were less intelligible than Lovelace's algorithm, bearing little relationship to any kind of written word. For example, the ENIAC, a fully electronic computer built at the University of Pennsylvania from 1943 to 1945, was programmed initially by plugging in an elaborate set of cables (da Cruz 2011). To run particular calculations, constants were then set using dials. Though the notion of punch cards dates back at least to Babbage in the nineteenth century and Falcon and his loom from the eighteenth century, programming the ENIAC was a matter of direct physical interaction with hardware rather than the manipulation of symbols. The operators of the ENIAC, who were primarily women, "played an important role in converting the ENIAC into a stored-program computer and in determining the trade-off between storing values and instruction" (Chun 2011, 31). Historically, even as programming continued to expand away from direct hardware manipulation and into progressively higher levels of abstraction, these operators were inventing both computation and the act of programming as embodied, materially engaged activities and vocations.

Machine Language and Assembly Language

The move beyond cables and dials was accomplished with machine language. A program in machine language is simply a sequence of numbers that causes the computer to operate but can be understood by humans. On the ENIAC, the numbers that formed a machine language program were decimal (base 10), but different bases were used on other early systems. The EDVAC used binary; the ORDVAC, octal; and the BRLESC, sexadecimal (Bergin 2000, 62). The numbers specify what operations the computer is to carry out and consist of *opcodes* (indicating low-level commands) that may be followed by *operands* (giving one or more parameters to those commands). For instance, the opcode to add a value to the accumulator has to have one operand after it, specifying what value is to be added, while the opcode to increment the x register does not have any operands at all, since that opcode by itself specifies everything that is to be done.

To jump unconditionally to a particular absolute address, an opcode such as "76" (in base 10) is used, followed by two bytes specifying the

```
{160} 10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

address. In fact, this is the opcode used for an unconditional branch to an absolute address on the Commodore 64. While the sequence of numbers in a machine language program is unambiguous to the computer, it is far from obvious at a glance even which numbers represent opcodes and which operands. An expert in machine language could pick out some patterns, but would often have to start at the beginning, recognizing each opcode, knowing how many operands correspond to those opcodes, and continuing through as if simulating the calculations and memory storage operations as the program executes. Writing a machine language program requires similar low-level expertise. Working at this level is clearly something that computers do better than people, as was acknowledged when programming took the next step.

A more legible form of code arose in the second generation of programming languages, called assembly languages. Assembly allows mnemonics for operators such as `lda` (load accumulator), `jmp` (jump), and `inc` (increment memory) to stand in for the more esoteric numerical codes. On the Commodore 64, the letters `jmp` followed by an absolute address are converted by the assembler to 76, translating the human-legible mnemonic into machine language. The first assembler ran on Cambridge University's EDSAC in 1949. EDSAC was, incidentally, the first computer capable of storing programs, a feature modern computer users have taken for granted for decades (Campbell-Kelly and Aspray 1996, 184). Although cryptic compared to a high-level language such as BASIC, an assembler program is nevertheless more comprehensible to a human than a machine language program. While assembly is still used today in, for instance, programming firmware and in the demoscene, there are usually significant disadvantages to programming at this level. While programming in assembly highlights technical details of platforms and the transfer of values between memory locations and registers, higher-level languages allow the programmer to concentrate on other matters, such as solving mathematical and scientific problems or modeling financial processes.

High-Level Languages

During the first decade of electronic computing, programming was still crawling toward the ideal of Lovelace's "program" that specified an algorithm in high-level, human-readable form. To reach a level of programming

more appropriate for mathematical and scientific tasks, FORTRAN (FORmula TRANslation) was outlined in 1954 and first implemented in 1957. This language, and particularly the next version of it (FORTRAN II, which appeared in 1958), had a very strong influence on BASIC. Just as FORTRAN was designed with mathematics in mind, COBOL (COMmon Business-Oriented Language) was introduced in 1960 to address business uses.

FORTRAN and COBOL both allowed for more intelligible code, improving on assembly. But both were also developed in the context of batch processing, for use with stacks of punched cards that would be processed by the machine one at a time. Punch cards were first used in the eighteenth century to define patterns in mechanical textile looms, as discussed in the chapter *Regularity*, but the concept was adopted in computing in the twentieth century. The paragon of the punched card became known as simply “the IBM card,” the eighty-column punched card introduced by that company in 1928. The Commodore 64’s eighty-column logical line, which appears as two forty-column lines on the screen, is one legacy of this early material medium for computing.

Programs written in early versions of COBOL and FORTRAN were specific to the punched card in definite ways. COBOL reserved columns 1–6 for the line number and column 7 for a continuation mark if the card’s code ran on from the previous one. FORTRAN was similar, with columns 1–5 indicating the statement number. “Comments” (usually the program name) went in columns 73–80 in both languages; a “C” in the first column indicated that the whole FORTRAN card was to be considered a comment. Unless a programmer wrote a one-liner—which in this case means writing a program that fits on a single card—a COBOL or FORTRAN program would take the form of a stack, often a massive stack, of punched cards. These had to be punched on a keypunch machine and fed into a card reader by an operator. Line numbers were essential in one particular case: if someone dropped a stack of cards and they needed to be sorted back into order.

FORTRAN’s `GOTO` command was the basis for the `GOTO` in the original BASIC (Kurtz 2009, 86), which was carried over into Commodore 64 BASIC. `GOTO` functions in the same way the assembly language `jmp` does, shifting the interpreter to a specific location within the program. If one were simply interested in easily writing `jmp` statements, BASIC offers little advantage. The benefits of BASIC can be seen in commands such as `PRINT`. What takes many steps in machine language is efficiently accomplished

```
{162} 10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

by this single command that employs a clear, natural language metaphor.

BASIC was a language specifically designed for the next computing revolution, one that would go beyond punched cards and batch processing to allow numerous users interactive access to a system simultaneously. This revolution was time-sharing.

Dartmouth BASIC and Time-Sharing Minicomputers

In 1962, change was sweeping through Dartmouth. Late that year, in an article that declared the isolated college was coming “out of the woods,” *Time* noted that the school had built a major new arts center and that John Kemeny, who had made full professor at age twenty-seven, had built “the best college math department in the country.” Also in 1962, Kemeny and his colleague Thomas Kurtz had begun developing a time-sharing computer system and a new programming language to be used by all students at Dartmouth—not just those studying math, science, or engineering. Kemeny and Kurtz aimed for nothing less than a computing revolution, radically increasing access to computers and to computer programming. “While the availability of FORTRAN extended computer usage from a handful of experts to thousands of scientific users,” Kemeny wrote, “we at Dartmouth envisaged the possibility of millions of people writing their own computer programs” (Kemeny 1972, 30).

To reach millions, Kemeny and Kurtz would have to lower the existing barriers to programming, barriers that were related not only to the esoteric aspects of programming languages, but also to the physical limits and material nature of computing at the time. The two began working with a team of undergraduates to develop the Dartmouth Time-Sharing System and, with it, the BASIC programming language (figure 50.1). They considered developing a subset of FORTRAN or Algol, but found these options unsuitable (Kurtz 2009, 79). As Kurtz told an interviewer, “we wanted . . . to get away from the requirements that punched cards imposed on users, which was that things had to be on certain columns on the card” (81). They saw the value of an interactive, time-sharing system for allowing users to correct minor errors quickly rather than coming back twenty-four hours later with a new stack of punched cards for their next scheduled batch job. They also relaxed some of the specific requirements that were tied to using keypunch machines and cards. Oddly enough, BASIC was so relaxed

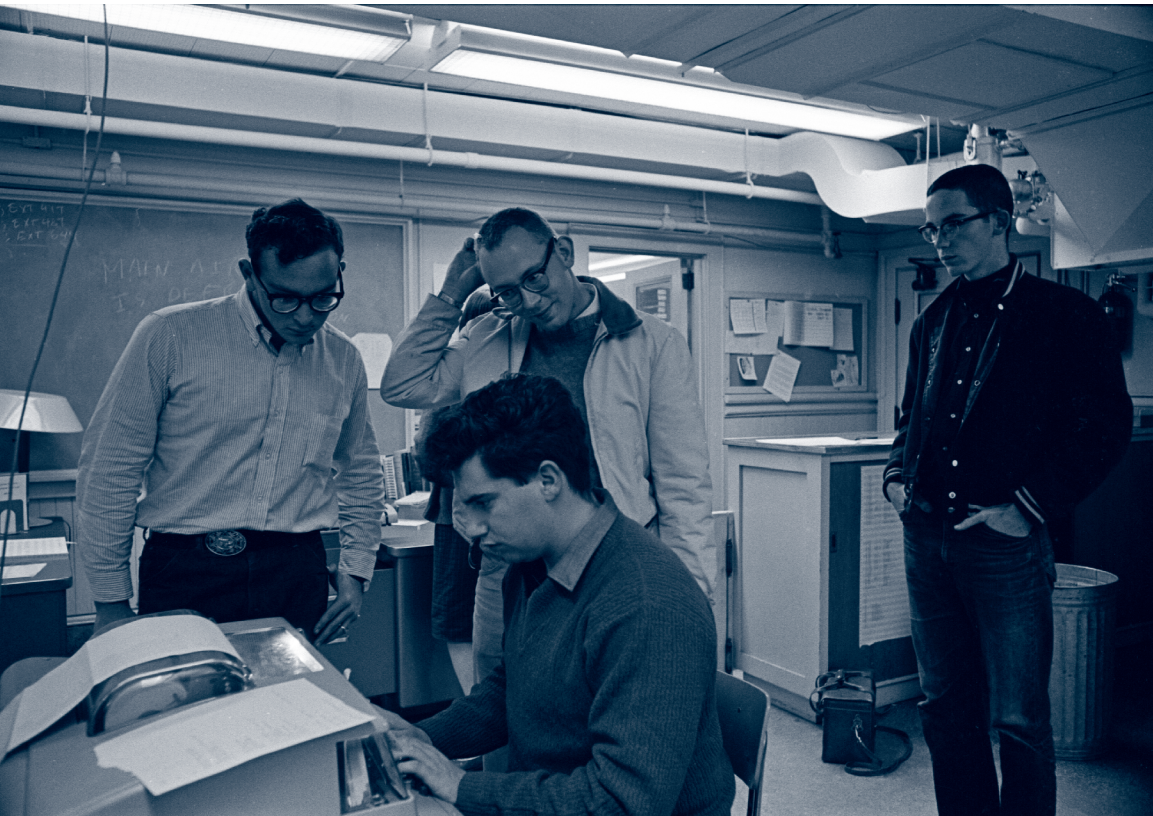


Figure 50.1

This image illustrated John Kemeny and Thomas Kurtz's essay "Bringing Up BASIC" with the caption "Students at Dartmouth working with the first version of BASIC." Photo by Adrian N. Bouchard, courtesy of Dartmouth College Library.

that spaces between tokens were optional in Dartmouth's versions of the language. Spaces were ignored, as Kurtz explains, because "some people, especially faculty members, couldn't type very well" (81). This aspect of Dartmouth BASIC was carried over onto the Commodore 64.

BASIC was designed in other ways to help new programmers, with "error messages that were clear and friendly" and default options that would satisfy most users' needs (Kemeny and Kurtz 1985, 9). "If the expert needs something fancier," the creators of the language declared, "let the expert do the extra work!" (11). Kemeny and Kurtz envisioned BASIC as a

```
{164} 10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```


true high-level language, allowing programmers to operate without any detailed knowledge of the specific hardware they were using. The initial idea, at least, was that programmers need only pay attention to BASIC instead of the computer that BASIC happened to be running on.

DTSS and the versions of BASIC that ran on it served almost all of the students at Dartmouth; by 1971, 90 percent of the seven most recent classes of freshmen had received computer training. Dartmouth extended access to its system to other campuses and also inspired the creation of other time-sharing systems with BASIC. By 1965, General Electric, on whose computers DTSS and the original BASIC ran, was offering a commercial time-sharing service that included BASIC (Waldrop 2001, 292). Well before the microcomputer revolution of the late 1970s, other college and university students were taking required courses in computing using BASIC—at NYU’s business school, for instance. Even before the arrival of home computers with built-in BASIC, the language was very widely used. The permissive attitude of Kemeny and Kurtz led to many different implementations of BASIC for different systems. Writing in 1975, one observer noted that “BASIC systems differ among themselves in much the same way that the English language as we know it differs among the English-speaking nations around the globe” (Mullish 1976, 6). There was a downside to this, however: the very permissiveness that led to BASIC’s widespread adoption and adaptation meant that the language, as actually implemented, wasn’t as independent of particular hardware as Kemeny and Kurtz had planned.

In addition to BASIC and the DTSS, there is yet another legacy from Dartmouth that has powerfully swayed the direction of modern computing: the almost evangelical mission to foster a more productive and creative relationship to computing. In his 1972 book *Man and the Computer*, Kemeny defends programming and playing games and other “recreational” uses of the computer as important, writing that such activities are relaxing and help people to overcome their fear of computers (35). Kemeny also describes some of the context in which BASIC programming was done at Dartmouth: “The computation center is run in a manner analogous to Dartmouth’s million-volume open-stack library. Just as any student may go in and browse the library or check out any book he wishes without asking for permission or explaining why he wants that particular book, he may use the computation center without asking permission or explaining why he is running a particular program” (33). Computers were for everyone (at

CHR\$ AND THE DOLLAR SIGN IN BASIC

In designing BASIC, Kemeny and Kurtz wanted to distinguish between variables that held numeric values and variables that held strings of text. They chose to have string variables and string functions end with a "\$," so that a string variable might be named A\$ and the function that produced one-character strings based on ASCII values was called CHR\$. They selected the dollar sign because there "were not so many keys on a Teletype, and we needed to find one that had not yet been used for BASIC. Of the few remaining ones, none seemed very appropriate. Then one of us observed that \$ looks like S for string" (Kemeny and Kurtz 1985, 28).

Computers are fundamentally machines that add and multiply, so it is a curious circumstance that the Commodore 64 keyboard, like modern North American keyboards, has a dollar sign on it but does not have a multiplication sign. Instead, the asterisk (*), the typographical mark once used to indicate a footnote, is pressed into service as the symbol for multiplication.

Why would a keyboard have a dollar sign but not a multiplication sign? Even though interactive processing (as opposed to batch processing with punched cards) was a novelty, teletypewriters had been used as interfaces to computers long before the 1960s. Various sorts of TTYs, teleprinters, or "printing telegraphs" were used commercially as a means of textual communication beginning in the 1910s. These

least within the campus community) and for any purpose. BASIC was the embodiment of this openness, which allowed for programs with no obvious military, business, or scientific purpose—programs such as **10 PRINT**—to come about.

It's not surprising that Kemeny's liberal ideas about computers and education played some part in his achievements as the president of Dartmouth College, a position he served in from 1970 to 1981. Kemeny presided over Dartmouth's conversion to a coeducational campus, removed the "Indian" as the college's mascot, and encouraged the recruitment of minority students. On his final day as president, he gave a commencement address that warned students, including those involved in the recently founded conservative *Dartmouth Review*, against the impulse that "tries to divide us by setting whites against blacks, by setting Christians against

systems were based on pre-ASCII character codes that came from telegraphy, and were used to transmit dollar amounts much more frequently than they were used for sending mathematical equations. Murray Code, the 1901 revision of the standard 1870 telegraph code, included two characters that allowed “shifting” into alternate sets. A shifted “D” would become a “\$” in Murray Code.

Interestingly, the original, North American Commodore 64 keyboard sported a pound sterling sign (£), a glyph absent from other US computers of the time. The presence of this key no doubt pointed to Commodore’s plans to sell its computers in the United Kingdom, although that key had a precedent in Murray Code, which also features a pound sterling sign.

The dollar sign is still used in some of today’s workhorse programming languages, such as Perl and PHP. In both of these, it is used to indicate a variable by being placed at the beginning, rather than the end, of the variable name. In Perl it specifically indicates a *scalar* variable, since \$ looks like S for “scalar.” Despite these varying uses, the impact of BASIC’s role as an entry-level language in the 1970s and 1980s was such that some modern programmers, including one of this book’s authors, still pronounce “\$” as “string” when reading code aloud regardless of the character’s meaning in the language under discussion.

Jews, by setting men against women. And if it succeeds in dividing us from our fellow beings, it will impose its evil will upon a fragmented society” (Faison 1992). After leaving the presidency of Dartmouth, he returned to full-time teaching. Kemeny died in 1992. Thomas Kurtz continued to teach at Dartmouth long after he worked on BASIC, directing the Computer and Information Systems program there from 1980 to 1988.

BASIC COMES TO THE HOME

The computer had already become more welcoming and accessible thanks to innovations on university and college campuses, but it wasn’t until the computer moved into the home that the true revolution began. In popular

lore that revolution started with a single photograph of a panel of switches and LEDs on the cover of the January 1975 issue of *Popular Electronics*—the first public look at the Altair 8800 minicomputer (figure 50.2). While the history of BASIC at Dartmouth shows that personal computing did not suddenly spring to life fully developed in 1974, that year does mark an inflection point for home computing.

Proclaiming in a headline that “THE HOME COMPUTER IS HERE,” that issue of *Popular Electronics* gushes about the possibilities of the \$395 Altair 8800 (\$495 assembled, \$1812 and \$2271 in 2012 dollars, respectively). The magazine claims that the computer can be used as a “sophisticated intrusion alarm system,” an “automatic IC tester,” an “autopilot for planes, boats, etc.,” a “time-share computer system,” a “brain for a robot,” and a “printed matter-to-Braille converter for the blind,” among other things, noting that “many of these applications can be performed simultaneously” (Roberts and Yates 1975, 38). As it happened, even the applications that were within the capabilities of the device were rather difficult to realize, since the system by default could only be programmed in machine language. Furthermore, unless one happened to have a Teletype or other terminal lying around, the programming had to be done using the toggle switches on the front panel. The home computer may have arrived, but most hobbyists would have no effective way of programming it. From the outset, it was clear to many that the Altair 8800 needed a programming language that facilitated experimentation. *Popular Electronics* mentions four programming languages by way of explaining the distinction between hardware and software (34); one of these languages was BASIC, the language that showed the most promise to early Altair 8800 enthusiasts.

Altair BASIC and Microsoft

There were two successful efforts to develop a BASIC interpreter for the Altair. Their varied histories have had a lasting impact on modern computing and modern culture. While this tale of two BASICs is not about the best of BASIC and the worst of BASIC, it does highlight two extremes in software development: one commercial, closely coordinated with hardware manufacturers and highly tied to licensing and cost structures; the other community based, nonprofit, and “copyleft.” The BASICs that led in these directions were Microsoft’s Altair BASIC, the official BASIC for the platform,

licensed to MITS, the Altair's manufacturer; and Tiny BASIC, which originated at the People's Computer Company or PCC. The Commodore 64's BASIC is directly descended from Altair BASIC, by the company whose name was later standardized as "Microsoft." The BASIC for this system is—although this is not visible on the startup screen—a Microsoft product. At the same time, the 10 PRINT program participated in a context of BASIC sharing and exploration that was strongly influenced by the People's Computer Company.

The story of Microsoft BASIC, and of Microsoft, begins with Paul Allen catching sight of the now-famous January 1975 issue of *Popular Electronics* just after it hit the stands. Allen read the Altair 8800 article and raced to show it to his friend and business partner Bill Gates, telling him, "Well here's our opportunity to do something with BASIC" (Wallace and Erikson 1992, 67). The two had already had some limited business success when Gates was still in high school, with a venture called Traf-O-Data that produced hardware and software to count cars. They programmed the Intel 8008 microprocessor that powered Traf-O-Data using a simulator running on a Washington State University mainframe.

By 1975, Gates was at Harvard University, and he and Allen were ready for a project with broader reach. Seeing *Popular Electronics*, they came up with the idea of writing BASIC for the Altair and called Ed Roberts at MITS, asking if he would be interested in having their BASIC for his system. Roberts said he would buy the first version that worked (Wallace and Erikson 1992, 74). As Gates and Allen had done when developing their Traf-O-Data system, they initially programmed on a university's computer system. They used a simulator to program the Altair 8800, a computer Gates and Allen had never seen in person. What they were programming at this point was, of course, a modified and minimized version of the original 1964 BASIC. Given how often the success of personal computing is attributed to entrepreneurial and business advances and to Microsoft in particular, it's remarkable that Microsoft's first product was developed by borrowing time on Harvard's computer and was (as Microsoft always acknowledged) an implementation of a freely available programming language from Dartmouth.

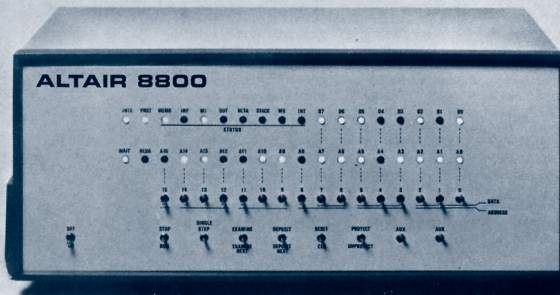
Gates and Allen devised the Altair BASIC project and did most of the programming, but there was a third Altair BASIC programmer, a sort of legendary "fifth Beatle." This was Monte Davidoff, who wrote the floating point routines. Gates and Allen were discussing how they needed to code

JANUARY, 1975

**EXCLUSIVE!**

ALTAIR 8800

The most powerful minicomputer project ever presented—can be built for under \$400



BY H. EDWARD ROBERTS AND WILLIAM YATES

THE era of the computer in every home—a favorite topic among science-fiction writers—has arrived! It's made possible by the POPULAR ELECTRONICS/MITS Altair 8800, a full-blown computer that can hold its own against sophisticated minicomputers now on the market. And it doesn't cost several thousand dollars. In fact, it's in a color TV-receiver's price class—under \$400 for a complete kit.

The Altair 8800 is not a "demonstrator" or souped-up calculator. It is the most powerful computer ever presented as a construction project in any electronics magazine. In many ways, it represents a revolutionary development in electronic design and thinking.

The Altair 8800 is a parallel 8-bit word/16-bit address computer with an instruction cycle time of 2 μ s. Its cen-

JANUARY 1975

tral processing unit is a new LSI chip that is many times more powerful than previous IC processors. It can accommodate 256 inputs and 256 outputs, all directly addressable, and has 78 basic machine instructions (as compared with 40 in the usual minicomputer). This means that you can write an extensive and detailed program. The basic computer has 256 words of memory, but it can be economically expanded for 65,000 words. Thus, with full expansion, up to 65,000 subroutines can all be going at the same time.

The basic computer is a complete system. The program can be entered via switches located on the front panel, providing a LED readout in binary format. The very-low-cost terminal presented in **POPULAR ELECTRONICS** last month can also be used.

PROCESSOR DESCRIPTION

Processor: 8 bit parallel
Max. memory: 65,000 words (all directly addressable)
Instruction cycle time: 2 μ s (min.)
Inputs and outputs: 256 (all directly addressable)
Number of basic machine instructions: 78 (181 with variants)
Add/subtract time: 2 μ s
Number of subroutine levels: 65,000
Interrupt structure: 8 hardware vectored levels plus software levels
Number of auxiliary registers: 8 plus stack pointer, program counter and accumulator
Memory type: semiconductor (dynamic or static RAM, ROM, PROM)
Memory access time: 850 ns static RAM; 420 or 150 ns dynamic RAM

33

Figure 50.2

The January 1975 issue of *Popular Electronics* featured the Altair 8800, which inspired the creation of Microsoft BASIC.

```
{170} 10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

these when Davidoff, a student sitting with them at the table, spoke up and said he could write them. After talking it over with him, they enlisted him to contribute them (Wallace and Erikson 76–77). The code of this first Microsoft project is mainly by Gates and Allen, though, with Gates listed as first author. Comments at the beginning of the code declare, “BILL GATES WROTE THE RUNTIME STUFF,” which he did over a period of eight weeks at Harvard. Gates would say years later that this 4 KB BASIC interpreter “was the coolest program I ever wrote” (76–77).

Allen flew to visit MITS in Albuquerque, taking along a paper tape with Altair BASIC on it. During the plane’s descent he realized that he did not have a way to get the Altair to read the tape and run it—a bootloader. So he wrote one in machine language as the plane was landing. He then went to see an actual Altair 8800 for the first time. The demo of BASIC, written on a simulator, was a success on the machine itself, and the interpreter was licensed by MITS. Gates and Allen moved to New Mexico to create new versions of BASIC for the Altair and to maintain the original code. Microsoft was booted and running.

Altair BASIC included alterations to Dartmouth BASIC, many of which would have made no sense on earlier time-sharing systems but which were helpful, even crucial, on home computers. While none of Microsoft’s changes to BASIC were critical to the functioning of `10 PRINT`, Gates and Allen did create the `POKE` and `PEEK` statements, which have been widely used in microcomputer BASIC and in programs found throughout this book.

`POKE` allows a programmer to write a value to a specific memory location. For example, `POKE 53272,23` places the value 23 into address 53272, a location in memory that is mapped to a register of the VIC-II graphics chip. In this case, `POKE 53272,23` switches the Commodore 64 into lowercase mode. `PEEK` is `POKE`’s complementary statement; instead of writing a value to memory, `PEEK` reads the value of a given location.

Both statements are extremely powerful. Altair BASIC—and later, the Microsoft BASIC used on the Commodore 64—sets no restriction on which memory addresses can be changed with `POKE`. This means, as the *Altair BASIC Reference Manual* warns, “Careless use of the `POKE` statement will probably cause you to ‘poke’ BASIC to death; that is, the machine will hang, and you will have to reload BASIC and will lose any program you had typed in” (1975, 35). This process would have been particularly painful on microcomputers on which BASIC was bootloaded rather than being

provided a part of the system's ROM. It's clear why earlier versions of BASIC did not include POKE or PEEK equivalents. A user on a time-sharing mini-computer should not have been able to write values directly to the micro-processor or memory; such a privilege would have threatened the stability of the entire shared system.

From a technical and business standpoint, Altair BASIC was not an early oddity, but rather, a Microsoft product with a strong relationship to the company's later flagship products. To see the connection, it's important to understand the nature of computing platforms and their relationship to markets of different sorts.

In *Invisible Engines*, Evans, Hagiu, and Schmalensee (2006) introduce a theory of two-sided software platforms. In a predominantly one-sided market—for example, a swap meet with people trading comics—there is only one class of participant, a person interested in exchanging goods with other people. A classic land-line telephone company also participates in a one-sided market, because every customer is more or less the same sort of participant, one who wants to make and receive calls. However, a credit card company has two different classes of customer: merchants, who receive payments and need terminals; and cardholders, who have a line of credit and make purchases.

When Atari released the Atari VCS in 1977, it was initially a one-sided platform. Atari made the system as well as all the games and controllers. The only participants in the market, and users of the platform, were the players. By the early 1980s, when Activision, Imagic, and other third-party companies had entered the market, there was another class of participant—one that was not paying royalties to Atari for the privilege of making games for their console.

Microsoft Windows is another example of a platform with at least two sides. On one side, computers need an operating system and desktop environment to function. This leads hardware manufacturers such as Dell to purchase licenses to Windows and to include the software with their systems. On the other side, computers are only valuable if there are applications written for them. Microsoft writes some of the applications for Windows, but third-party developers write many others. The abundance of software has a network effect that is positive for Microsoft: it encourages users to stay with Windows. And, since Windows is pre-installed on many computers, companies want to write applications for it.

Of course, Windows was not the first two-sided platform, or even Microsoft's first. By retaining the rights to what IBM called PC-DOS, Microsoft had previously been able to license MS-DOS to other companies, much as it would later sell OEM copies of Windows to them. And before that, there are continuities with the company's first product line, Microsoft BASIC. BASIC was a programming language, not an operating system, but the presence of BASIC allowed programs to be written on a computer and sometimes sold. There were at least two sides to BASIC as a software platform: the computer companies, beginning with MITS, who wanted it on their machines; and the computer hobbyists who wanted to write (and in many cases sell) BASIC programs. At Gates and Allen's young company, the success of BASIC and the essential business plan used with that family of software products formed the basis of Microsoft's later success licensing MS-DOS and Windows.

Tiny BASIC and Copyleft

Even as Microsoft was securing its future as a multisided company, addressing both manufacturer and computer user demand, a different version of BASIC—indeed, a different philosophy of software altogether—was brewing in the San Francisco Bay area. In 1975, volunteer programmers and the nonprofit People's Computer Company (PCC) developed an alternative BASIC for the Altair 8800. Bob Albrecht, who had founded “probably the world's first completely free, walk-in, public computer center—People's Computer Center—in a storefront in Menlo Park, California” (Swaine 2006) was one of many, along with Paul Allen, who had seen the *Popular Electronics* cover story on the MITS Altair. He discussed it with Dennis Allison, who taught at Stanford, and Allison began to develop a specification for a limited BASIC interpreter called Tiny BASIC. In a collaborative hobbyist spirit, Allison's documents were published in three parts by Albrecht in issues of the PCC newsletter, a serial that had been running since October 1972. At the conclusion of this series of specifications, Allison called for programmers to send in their implementations and offered to circulate them to anyone who sent a self-addressed, stamped envelope.

The first interpreter written in response to this call was by Dick Whipple and John Arnold and was developed in December 1975. To disseminate it, Albrecht and Allison started a new serial, initially photocopied and

originally intended to just run for a few issues. This was *Dr. Dobb's Journal of Tiny BASIC Calisthenics and Orthodontia*; it printed the code for the interpreter in octal machine language, ready for hobbyists to toggle in or, even better, key in on their Teletypes. It is an understatement to call this publication a success. By January of 1976 the journal title was made more general by removing the explicit mention of Tiny BASIC, an editor was hired, and *Dr. Dobb's* was launched as a newsletter offering code and articles on computing topics. In 1985, *Dr. Dobb's* further participated in the culture of sharing and openness by publishing Richard Stallman's "GNU Manifesto," a foundational document of the free software movement. The journal ran as a print periodical until 2009, with a circulation of 120,000 shortly before that. It still exists as an online publication.

The development of Tiny BASICs continued after Allison's first version. The fourth Tiny BASIC, written by Li-Chen Wang, was called Palo Alto Tiny BASIC. It, too, was published initially in *Dr. Dobb's*. The source listing for this BASIC interpreter began:

```
;*****
;*
;*          TINY BASIC FOR INTEL 8080
;*          VERSION 1.0
;*          BY LI-CHEN WANG
;*          10 JUNE, 1976
;*          @COPYLEFT
;*          ALL WRONGS RESERVED
;*
;*****
```

While this header does not use "copyleft" in the same sense that free software licenses would begin in the late 1980s, this anticopyright notice was a jab at the closed culture of locked-up, proprietary code. Because Wang chose to disclaim copyright and reserve only the "wrongs" of the program, Palo Alto Tiny BASIC was able to serve as the basis for a commercial BASIC: Level I BASIC for the TRS-80, the influential microcomputer that came on the market in late 1978.

```
{174} 10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

The Ethic vs. The Corporation

In *Hackers: Heroes of the Computer Revolution*, Steven Levy describes the early history of home computing and the development of BASICs for the Altair and its immediate successors as a battle between an ethic of openness and the sort of corporate powers who were “a foe of the Hacker Ethic” (1984, 227).

This portrayal has helped to set up what is often remembered as the first major clash between free software and the corporate will of Microsoft: Bill Gates’s “Open Letter to Hobbyists” (Gates 1976a). Published at the beginning of 1976 in numerous newsletters, including the *Altair Users’ Newsletter* and that of the Homebrew Computer Club, this confrontational letter gave Gates the opportunity to scold home computer users for the same kind of sharing that the original BASIC at Dartmouth encouraged. In the letter, Gates soundly declared, “As the majority of hobbyists must be aware, most of you steal your software.” The letter spurred hundreds of responses, public and personal, causing the first major controversy in home computing. Many see it as the start of Microsoft’s history of unfair dealing—and of embracing and extending, a practice in which the company takes existing tools and ideas and creates its own version for competitive advantage. Decades later, in the antitrust case against Microsoft, Judge Thomas Penfield Jackson would write that “Microsoft is a company with an institutional disdain for both the truth and for rules of law that lesser entities must respect.” He was hardly the only one with this view by that point. Did Microsoft’s rise to the height of corporate ruthlessness begin with this letter to hobbyists?

There are a few complications to the most popular version of this early clash, just as there would be complications in considering Altair BASIC as completely wrongheaded and Tiny BASIC as perfect in every way. To begin with, neither Microsoft, nor Gates, nor Allen was the party fingered in Levy’s book as the “foe” of the hacker ethic. That distinction belongs to Ed Roberts, the head of MITS, who was running a business with dozens of employees, many of whom would be furiously working at any given hour to fulfill computer hobbyists’ orders (and their dreams of computer ownership). At the beginning of 1976, “Micro-Soft” (the spelling had not been regularized by the time of the first letter) was simply two partners, both recent college dropouts, one a teenager and the other only slightly older.

LINE NUMBERS AND COLONS: RESPONSES TO CHANGING HARDWARE

Microsoft has used its position as the major implementor of microcomputer BASIC to make many changes to the language, adding and removing components that became necessary or obsolete as hardware progressed. For example, to allow programs to be written more compactly—an important feature given the Altair's switch and toggle interface, which could literally be painful to programmers—Microsoft's Altair BASIC introduced the colon (:) to place separate statements on the same line. Used this way, the colon allowed lines such as:

```
160 R=16:PRINT"HOW MANY DECKS (1-4)";
```

Multiple statements on the same line were not possible with minicomputer BASICs and the ANSI standard; Kemeny and Kurtz saw the compression of statements as potentially confusing rather than helpful. When they released their much-revised True BASIC for home computers in 1983, they still did not allow multistatement lines. The colon is, however, important in **10 PRINT**; this program could not be written as a concise one-liner without it.

Another prominent feature of **10 PRINT** is also an artifact of the hardware underlying the language. Unlike its punch card-derived predecessors, BASIC didn't require programmers to put anything in certain columns; yet there was a requirement that a number such as 10 appear at the beginning of each line. While line numbers made branching possible by allowing **GOTO** and **GOSUB**, BASIC did not technically need line numbers for this purpose. Labels, such as those used in preexisting languages like assembly, would have sufficed. The line number's real value is seen on a Teletype or other print terminal, or in any environment where full-screen, nonlinear editing is not an option. Line numbers make it easy in those cases to delete existing lines: simply type the line number without any further instructions, and the line disappears. To correct a single existing line, just retype a line with the same line number. To see what code is currently on a particular line, **LIST 50** will do the trick—although if one had written the line recently, one could also look up or literally “scroll” back to where the line had been printed out.

The practice of numbering program statements by multiples of five or ten is not merely rhythmic or aesthetic (as it is in this book's table of contents); the insertion of new lines in a program requires that, as an early manual puts it, “the original line

```
{176} 10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

numbers not be consecutive numbers” (Dartmouth College Computation Center 1964, 22). Numbering a program 10, 20, 30, and so on ensured that, between every existing program statement, there was room for nine more. It was an acknowledgment that a program is dynamic, rather than fixed and perfect. The *Commodore 64 User’s Guide* also advises that “it is good programming practice to number lines in increments of 10—in case you need to insert some statements later on” (33). While line numbers became an iconic feature of BASIC early in the personal computing era, they also contributed to a perception among programmers that the language is limited to simple tasks. Leaving space to add nine new lines of code between every original line may initially seem like plenty of flexibility, but sometimes, when programmers have a complex problem to solve, it is not enough. BASIC can still accommodate this situation by allowing the programmer to use `GOTO` and `GOSUB` in order to jump to as-of-yet unused numbers for a separate routine and then return to the original program flow when complete. Unfortunately, too many subroutines can result in “spaghetti code”—so named because the flow chart of a program becomes so confused and self-referential that all the lines look like a plate of spaghetti, making the program nearly unintelligible.

In 1991 Microsoft realized that the perception of BASIC as limited to simple programs was holding the language back, and that this perception was largely due to a feature of the language that was no longer even necessary. Text editors that could display and allow access to screenfuls, or windowfuls, of lines had long been available, and line numbers were now doing more damage than good. QBasic, released in 1991, was a Microsoft BASIC that deprecated line numbers, encouraging the use of assembly-like text labels instead.

That `10 PRINT` includes both the colon and the line number—features that have been added and removed from BASIC in response to hardware changes—signals that `10 PRINT` is from a particular time and related to a specific era of computing caught between competing input mechanisms. Its heritage and provenance are written right into its code.

The company had paid other people, such as BASIC contributor Davidoff, but would not get its first official employee until April 1976.

Furthermore, copyright protection for computer programs, on which Gates based his argument, was well established by 1976. While sharing of programs certainly happened in users' groups and other contexts, today's concept of free software had not been articulated at that time. Richard Stallman started the GNU project in 1983; he published the GNU Manifesto and founded the Free Software Foundation in 1985, a decade after Altair BASIC. An argument has been made that, even though the discussions of this period have been overlooked, some of the ideas important to the free software movement were first publicly stated in the columns of magazines and newsletters in response to Gates's letter (Driscoll 2011). But many hobbyists were not interested in free (as in freedom) software as it is conceptualized today; rather, they were interested in (as the editorial in the first issue of *Dr. Dobbs* explained) "free and very inexpensive" software.

In the aftermath of his first letter, Gates wrote a "Second and Final Letter," replying to objections raised by his readers. In it, he conceded that, at least for certain types of programs, the free sharing of code was likely to become the norm. He also suggested that good "compilers and interpreters" (such as Microsoft's Altair BASIC) would enable such shared software:

In discussing software, I don't want to leave out the most important aspect, vis., the exchange of those programs less complex than interpreters or compilers that can be written by hobbyists and shared at little or no cost. I think in the foreseeable future, literally thousands of such programs will be available through user libraries. The availability of standardized compilers and interpreters will have a major impact on how quickly these libraries develop and how useful they are. (Gates 1976b)

Gates certainly had a concept of software that would allow for it to be sold and tightly controlled by a corporation, but he was hardly seeking to eliminate hobbyist programming and the sharing of code. The company's policies did run counter to the ethic of the Homebrew Computer Club and the People's Computer Company in significant ways; yet Microsoft facilitated not only the creation of new software with its version of BASIC, but also the exchange of programs that Gates mentioned. Through licensing deals with computer companies, Microsoft did a great deal to bring BASIC

out of the minicomputer time-sharing environment and onto microcomputers and beyond, to early adopters and enthusiast programmers. Microsoft signaled it was not completely at odds with the PCC in Appendix M of the original Altair BASIC manual. It lists “a few of the many texts that may be helpful in learning BASIC,” all but one of which can be ordered from the PCC, whose address is also provided. Five of the six books are about BASIC specifically, but the manual also lists the radical and countercultural *Computer Lib/Dream Machines* by Theodore H. Nelson (1974), an edition of which Microsoft would itself publish in 1987.

Ultimately, BASIC became what it was in 1982 thanks to the institution of higher education where it was first developed, the corporation that implemented and licensed it for use on home computers (including the Commodore 64), and, significantly, the hacker ethic of code sharing that allowed BASIC programs—such as 10 PRINT—to circulate freely.

COMMODORE BASIC

The ability to program a computer—to use its general power in customized ways—was a core selling point for many home computers including the Commodore 64. Home computers were often positioned against videogame systems in advertisements. Implicitly, this comparison reminded the prospective buyer that a computer could be used to play video games; explicitly, it pointed out that computers could be used with business and educational software—and that they could be programmed to do much more. This point was driven home in the many Commodore TV ads that compared the VIC-20 to game systems, including one in which William Shatner says “unlike games, it has a real computer keyboard” (Commodore Computer Club 2010).

That computers were programmable and that they specifically could be programmed in BASIC were hardly afterthoughts in their development or marketing. A Commodore 64 advertisement that was aired in Australia in 1985 provides evidence that BASIC was a central selling point (Holmes3000 2006). After the television spot showed bikini-clad women descending a waterslide (“♪ In a world of fun and fantasy . . . ♪”) and cut to a woman happily using a Commodore 64 in a retail store (“♪ . . . and ever-changing views . . . ♪”), it cut once again: to a screenful of BASIC, and then to depict a boy

programming in BASIC ("♪ . . . and computer terminology . . . Commodore and you! ♪"). The commercial suggests that computer programming was an obvious, important, and fun use of a home computer.

An early print ad for the Apple II that ran in *Scientific American* among other publications boasted, "It's the first personal computer with a fast version of BASIC—the English-like programming language—permanently built in. That means you can begin running your Apple II the first evening, entering your own instructions and watching them work, even if you've had no previous computer experience." It was very easy for home computers users to type in or modify a BASIC program, and the fact that the manufacturers encouraged such behavior in mass media advertising primed users to partake of programming once they'd purchased a machine.

At the opposite extreme were programs fixed in the ROM of cartridges, such as the cartridges of videogame systems. They were convenient, and they showed that such game systems had the flexibility to work in many different ways, but hacking cartridge code or writing one's own programs on a cartridge-based system was far from easy in the early 1980s. The Commodore 64 provided the flexibility of BASIC out of the box, but—like the TI-99/4A, among other computers—it also had a cartridge slot. By offering BASIC along with the ability to plug in cartridges (many of which were games), the Commodore 64 turned one of its Janus-like faces to the generality and power of home computing and another to the convenience and modularity of gaming.

The Commodore 64 BASIC on which **10 PRINT** runs is a Microsoft product and a descendant of Altair BASIC. The first step for achieving this BASIC was creating a version for the Commodore 64's chip, the MOS 6502 processor. The Altair had used the Intel 8080, which had a different instruction set. The task of developing a version of Microsoft BASIC to work with the MOS 6502 was undertaken in 1976 and fell mainly to Richard Weiland. When a user types "A" in response to the startup question "MEMORY SIZE?", the version of Microsoft 6520 BASIC licensed to Ohio Scientific replies "WRITTEN BY RICHARD W. WEILAND," while version 1.1 for the KIM declares "WRITTEN BY WEILAND & GATES." The first version of Microsoft's 6502 BASIC that made its way into the ROM of a shipping system, in 1977, was a version for Commodore—not for the Commodore 64, but for the company's first computer, the Personal Electronic Transactor or PET.

The BASIC included with the Commodore PET was very similar to

```
{180} 10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

Commodore BASIC 2 for the Commodore CBM and the BASICs included on the VIC-20 and Commodore 64. (The version history of Commodore BASIC is a bit complicated, as the “COMMODORE 64 BASIC V2” that appears on the top of the screen indicates the second release of BASIC V2.0; this version was originally provided with a model of the PET 2001.) Other aspects of Commodore computers, such as the PETSCII character set, are similar across models as well. For these reasons, **10 PRINT** will run without modification on a Commodore PET or a VIC-20. What mainly suggests that the program should be identified with the Commodore 64 is the presence of **10 PRINT** variants in a Commodore 64 manual, a later magazine, and other contexts specific to the Commodore 64.

Versions of Microsoft’s 6502 BASIC were used not only on the PET and Commodore 64 but also on competing computers: the Apple II series and Atari’s eight-bit computers, the Atari 400 and Atari 800. Microsoft certainly benefited from selling the same product to multiple computer manufacturers but didn’t manage to make the usual licensing deal with Commodore. As the founder of Commodore, Jack Tramiel, explained at the Computer History Museum at a twenty-fifth anniversary event honoring the Commodore 64, Bill Gates “came to see me. He tried to sell me BASIC. And he told me that I don’t have to give him any money. I only have to give him \$3 per unit. And I told him that I’m already married.” Tramiel told Gates he would pay no more than a \$25,000 flat fee for BASIC—an offer that Microsoft ultimately accepted. This was a very good deal for Commodore, since about 12.5 million Commodore 64s were ultimately sold (Steil).

Features of BASIC highlighted by **10 PRINT** and which are fairly specific to the Commodore 64 version are seen in the RND command, discussed in the Randomness chapter, and in the PETSCII character set that CHR\$ refers to, discussed in the Commodore 64 chapter.

Before turning to the way Commodore 64 BASIC programs circulated, it’s worth noting what the creators of Commodore 64 BASIC went on to do. Bill Gates’s career trajectory is, of course, well known. It is indeed the case that one of the programmers of Commodore 64 BASIC became the richest person in the world. Paul Allen is not far behind in wealth; he left full-time work at Microsoft in 1982. Gates and Allen are notable philanthropists today. The other coder who contributed to the original Altair BASIC, Monte Davidoff, did not strike it nearly as rich but, according to an interview, was still active as a programmer in 2001, running Linux and preferring

to program in Python (Orlowski 2001).

The programmer of Microsoft's 6502 BASIC, Richard Weiland, a grade-school classmate of Gates and Allen, joined Microsoft when the company was based in Albuquerque. He worked for Microsoft until 1988 and devoted much of his time and money from then until his death in 2006 to philanthropy. He supported the Pride Foundation (with the largest-ever donation to LGBT rights), Stanford University (with the largest donation that university had ever received), the Audubon Society, and the Nature Conservancy (Heim 2008). While the dichotomy between profit-driven corporations and people-powered programming is not artificial, the generosity of the people behind Commodore 64 BASIC shows that those on the corporate side aren't without altruistic, community concerns.

THE CIRCULATION OF BASIC PROGRAMS

From the first years of the language, BASIC programs circulated as ink on paper. In 1964 and for many years afterward, there was no Web or even Internet to allow for the digital exchange of programs, and it was often impractical to distribute software on computer-readable media such as paper tape. From the mid-1970s through the early 1980s, BASIC was known in print not only through manuals and textbooks that explicitly taught programming, but also through collections of programs that appeared in magazines and computer club newsletters. The printed materials that remain today reveal insights into the practices of BASIC users and the culture that developed among them.

Programs in Print

Computer magazines often featured BASIC programs a home user could easily key in to his or her home computer. There was the previously mentioned *Dr. Dobb's*, but also many others. For instance, *Creative Computing* was a significant early magazine for microcomputer hobbyists. Launched in 1974 before the debut of the Altair 8800, *Creative Computing* was published until 1985 and spanned the era of BASIC's greatest growth and popularity.

As *Creative Computing* was nearing the end of its run, other mag-

```
{182} 10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

azines, many of them platform-specific, were just getting started. One was the previously-mentioned *RUN* (1984–1992), a monthly magazine published by IDG Communications, focused on the Commodore 64 and VIC-20. *RUN* is particularly noteworthy in the current discussion because a one-line variant of **10 PRINT** appeared in its pages in 1984. A German edition of *RUN* was published as well, and *ReRUN* disks made programs in the magazine available for purchase in machine-readable format. Another home computer magazine of this era was *Compute!* (1979–1994), which began as *Pet Gazette*, a 1978 magazine put together by Len Lindsay about the Commodore PET computer. In July 1983, *Compute!* launched a spinoff publication, *Compute!’s Gazette*, for Commodore 64 and VIC-20 owners. In the UK there were several magazines for Commodore users, including *Zzap! 64*, *Commodore User*, and *Commodore Format*—suggesting that putting the pound sterling symbol on the Commodore keyboard was a good move after all. Interestingly, the last of these UK magazines did not start publishing until October 1990, well into the twilight of the Commodore 64. *Commodore Format* was not about BASIC or learning to program, however, instead focusing on what was by 1990 nearly retro-gaming. The magazine came with a “Powerpack” tape, offering full games and demos for subscribers to load and run.

While magazines were ready and regular sources of BASIC programs, many enthusiasts also discovered code in long-form books. David H. Ahl’s influential compilation, *101 Basic Computer Games*, was first published in 1973 by the Digital Equipment Corporation (DEC). This was the first book to collect only games in BASIC. It includes a sample run of each game and acknowledges the programmers who contributed them. Each program’s “computer limitations” are described so that users understand the specific BASIC dialects and hardware that are supported, evidence that even as early as 1973 BASIC had drifted from its creators’ goal of a platform-agnostic high-level language. Hand-drawn illustrations punctuate *101 Basic Computer Games*—a rather playful presentation for a corporate publication. The game’s titles are all abbreviated so they can serve as filenames, which at the time were limited, on some systems, to six characters. The abbreviations are humorously cryptic, with ACEYDU standing for “Acey Deucy Card Game”; AMAZIN for “Draw a Maze”; or POET for “Random Poetry (Haiku).” In the preface, the educational value of playing and creating games and the need for “unguided learning” are emphasized, echoing

Kemeny's own thoughts about the value of play on computers.

A new edition of this book, *Basic Computer Games, Microcomputer Edition*, appeared in January 1978—reflecting BASIC's move from time-sharing minicomputers to microcomputers. This edition's preface begins with a dictionary definition (from an unnamed dictionary) of the word "game." It then provides a cursory history of sports and games from ancient times to the modern age, emphasizing that games offer recreational breaks from the "realities of life" and have other "important redeeming virtues." The programs in this book are meant to run on the gold standard of microcomputer BASICs: MITS Altair 8K BASIC, Rev. 4.0 (ix). The games, no longer referred to by cryptic six-character tags, are organized by category—e.g., Educational, Number and Letter Guessing, Sports Simulation, Gambling Casino, Card and Board, and so on. While none of these categories would easily accommodate **10 PRINT**, it is notable that so many of them rely upon a key feature of that program: randomness.

Later in 1978, this compilation was published again by Workman Publishing under the lengthy title *Basic Computer Games, Microcomputer Edition. 101 Great Games to Play on Your Home Computer. By Yourself or with Others. Each Complete with Programming and Sample Run*. Its translation into German in 1982 (reprinted in German in 1983) shows how BASIC games, thanks in this case to a book's large trade publisher, made their way abroad.

The People's Computer Company not only published a newsletter (figure 50.3) but also offered a book collecting BASIC games: Bob Albrecht's large-format *What to Do after You Hit Return* came out originally in 1975. This popular book underwent several printings from different publishers. Not once did the acronym BASIC appear on the front or back cover, perhaps indicating that the language was so prevalent for recreational programming that it need not be named.

Given the growing popularity of BASIC and computers among hobbyists, it is not surprising to see books of BASIC that go beyond games. Promising to teach a BASIC that would work with all the various "dialects," the 1977 *Illustrating BASIC (A Simple Programming Language)*, was published by no less a scholarly authority than Cambridge University Press. In 1978 came *The Little Book of BASIC Style: How to Write a Program You Can Read*, by John M. Nevison. With its allusion to the *Elements of Style* by Strunk and White, this book insists that programs have human readers and can be written with them in mind. Similar titles include the 1978 BASIC

with Style. *Programming Proverbs* and the 1979 *Programming for Poets. A Gentle Introduction Using BASIC*. Lest the utilitarian function of computers become overshadowed by these more aesthetically oriented books, there is Charles D. Sternberg's *BASIC Computer Programs for the Home* (1980), filled with programs specifically designed to satisfy "the practical requirements of the home." Should one of these programs not work on the reader's own machine, Sternberg encouraged their "modification."

This quick survey of BASIC books from the 1970s and early 1980s highlights the extent to which BASIC facilitated exploration, play, modification, and learning. It also reveals the nature of the home computing movement at the time, which emphasized sharing and learning from others, often through the medium of print. While programs in machine language occasionally circulated in print, published BASIC programs such as **10 PRINT** were a different beast altogether. BASIC was legible code. It could be read straight from the page and some sense of the program's nature was evident before the program was ever even executed. Furthermore, as a user typed in a program, he or she could easily alter it, sometimes mistakenly, yet often with purpose. Sometimes the magazines and books had typos themselves or didn't work with a particular reader's dialect of BASIC, and modifying the program—debugging it—became essential. The transmission of BASIC programs in print wasn't a flawless, smooth system, but it did encourage engagement with code, an awareness of how code functioned, and a realization that code could be revised and reworked, the raw material of a programmer's vision.

BASIC in Human Memory

Not so long ago, software was primarily transmitted on physical media, such as cassette tapes, floppy disks, and CD-ROMs. The notion that programs would routinely be published in print and typed in by users seems alien now. But there is an even stranger way that programs, particularly short ones such as BASIC one-liners could make their way from computer to computer and from person to person: as memorized pieces of code, like a software virus whose host is a human rather than a machine.

The dream of total recall of a computer program appears in science fiction. In Cory Doctorow's short story "0wnz0red," a programmer named Murray spirals downward in terms of his life and his code quality after the

apparent death of his only friend (Doctorow 2002). When this friend returns, having actually hacked his own body and mind into near-perfection, Murray attains similarly superhuman capabilities. Among these is the ability to precisely remember large amounts of text, including technical documentation of code, which “he closed his eyes and recalled, with perfect clarity.” Murray’s powers of recall even extend beyond human language. As the story ends, Murray “had the laptop open and began to rekey the entire codebase, the eidetic rush of perfect memory dispelled all his nervousness, leaving him cool and calm as the sun set over the Mission.”

In Doctorow’s story, the ability to memorize a large program is a superpower. At the same time, the story’s treatment of the human body and mind as machines that can be mastered by a programmer and owned (or even *Ownz0red*) by an adversary is consistent with the idea that memorizing code is enslaving one mind’s to the machine, treating it as a storage peripheral. For many programmers, however, memorizing one-liners (not an entire massive codebase) is both possible and useful—and pleasing, much as memorizing a poem might be. Such memorization would hardly seem strange in the context of what N. Katherine Hayles (2005) calls the “legacy systems,” speech and writing, out of which code evolves (as developed in her chapter “Speech, Writing Code: Three Worldviews”).

In the case of home computing in the early 1980s, it could be advantageous for someone to memorize a one-liner or a handful of short programs. A memorized one-liner could be typed into a friend’s computer, initiating a kind of two-way cultural economy; in exchange for sharing a particularly interesting or visually affecting program, one earned prestige, or street cred, a currency a teenager in the early 1980s would surely appreciate. In the late 1970s and early 1980s, many of these short programs worth memorizing were of course in BASIC. (See the remark *One-Liners* for several examples.) And because programmers typically had to memorize certain BASIC statements anyway, such as `? CHR$(147)` (which clears the screen), it was not much of a leap to memorize a program that contained two statements in that language, such as `10 PRINT CHR$(205.5+RND(1)); : GOTO 10`.

In addition to being able to show off and seem elite, there are some strictly utilitarian reasons to memorize a short program. For example, Perl is used for a wide variety of text-processing tasks; many who program in it find it useful to write, recall, or adapt one-line programs that work on the command line. For instance, to convert a file `dosfile.txt` with DOS-

STUDIES OF PROGRAM MEMORIZATION

In Doctorow's story, Murray finds it easier to remember pages of code than a string of random characters, an idea supported by experimental research. In an effort to better understand how people comprehend computer programs—which has ramifications for programmer efficiency—scientists began studying program memorization in the mid-1970s. In 1976 Ben Shneiderman, referring to the cognitive science literature on remembering sentences, reported a statistically significant difference between subjects of different ability levels attempting to memorize a FORTRAN program and a series of valid statements in scrambled order, “leading us to the conclusion that the structure of a program facilitates comprehension and memorization” (1976, 127).

Ruven Brooks published a more elaborate theory of program comprehension in 1983; **10 PRINT** shows some ways in which this theory is not generally applicable. In Brooks's theory, a programmer reconstructs knowledge about the real-world domains that the program models, developing an initially underspecified hypothesis and refining and elaborating it “based on information extracted from the program text and other documentation” (1983, 543). Brooks posited the useful idea that certain lines were “beacons,” indicating key program structures and operations (548). But his theory is otherwise difficult to apply to **10 PRINT**. Brooks lists seven internal and five external “indicators for the meaning of a program,” including “Indentation or pretty-printing” and “Flowcharts” (551). None are present in **10 PRINT**. More fundamentally, Brooks's model is meant for programs that simulate recognizable business processes, not computer programs in general. In fact, Brooks's model cannot directly apply to any creative program that lacks a real-world domain.

Brooks's concept of beacons was revisited in another memorization study by Susan Widenbeck. In this 1986 study, Widenbeck found that beacon lines were recalled more often by experienced programmers, presumably because they know what to look for. Widenbeck also noted, “Memorizing a program is a very unusual programming task, and it is possible that it changed the subjects' normal program comprehension strategies and procedures” (1986, 705). Interestingly, given the contemporary mantra that “code is poetry,” Widenbeck found that program memorization was not at all like the memorization of text: “If subjects were taking a strictly linear, or text-reading, approach to understanding the program, we would expect the lines near the beginning of the program to be better remembered . . . this was

{188} 10 PRINT CHR\$(205.5+RND(1)); : GOTO 10

not the case" (707). At the same time, Widenbeck made analogies to the reading of texts in describing how her subjects read programs: "Using beacons to understand a program seems to be something like skimming an English text. They help to figure out the general, high-level function, but they do not contribute to a detailed understanding of the code" (708).

Shneiderman's early acknowledgement that programs can be memorized is significant; his result, furthermore, shows that the memorization of programs relates to the memorization of language in some important ways (Shneiderman 1976). But there are differences between this study's perspective and the consideration of how BASIC one-liners are memorized. Memorization was used as a barometer of comprehension, the real focus of this research. Although it seems evident from these studies that memorization and comprehension are deeply connected, memorization is nevertheless being considered along the way to understanding something else. Shneiderman's study is about the short-term memorization of programs of about twenty lines, assigned as a task. Because short-term memory is the concern, Shneiderman cites Miller's famous paper on the topic, "The magical Number Seven, Plus or Minus Two" (Miller 1996) and discusses how his results may be consistent with the ones in that paper.

10 PRINT represents a category of programs that were historically memorized but that existing theories of program memorization, formulated for longer programs that model business processes, do not cover. The memorization of BASIC one-liners was rather long-term, done for fun, and of course involved very short programs. This type of memorization has certain things in common with the tasks done in these memorization studies, but it may also relate to the way people memorize jokes, proverbs, and other oral texts. These studies do at least show how people can remember key statements (beacons) and the high-level workings of a program without memorizing it character by character. Also, the appearance of program variants that work identically but are written differently is consistent with studies on and theories of program memorization.

style line endings to Unix format, a common task for many programmers, the carriage returns (indicated `\r`) must be removed. This can be accomplished with the command `perl -p -i -e 's/\r\n/\n/g' dosfile.txt` which includes a very short program (between quotes) to perform the needed substitution. With a few changes, this code can be adapted to replace one word with another, for instance, substituting “Commodore” for every occurrence of “Atari”: `perl -p -i -e 's/Atari/Commodore/g' manuscript.txt`.

Programmers who use such Perl one-liners do not seem to remember them in exactly the way one memorizes lines from a play or a song. They would generally understand how the substitution operator (`s///`) functions and how command-line flags to Perl work. In other words it is knowing Perl, not just the memorization of a string of symbols, that is important to most uses of Perl one-liners. But the phrase “Perl pie” (a mnemonic for `perl -p -i -e`) does help some to quickly recall which command-line flags are needed in this case, and one-liners are at times as much recalled as figured out and programmed. Many common one-liners are not programmed “from scratch” each time they are used.

This type of non-rote memorization is the sort that BASIC programmers also employed in bringing `10 PRINT` from one computer to another as they showed off its output. Remembering code, like having it printed with the occasional typo, was a “lossy” way to transmit programs. This didn’t have purely negative effects, though. Instead of the perfect but opaque way of transferring files via disk or download, the recall and reading of programs left a space for the programmer to work and play. If the recalled version of the program didn’t work correctly, the programmer could think about how to change it. If it did work well, the programmer still might think to change it, to see what else the program could do. The wiring of these printed and memorized programs was sometimes messed up, but they were not sealed from view in a closed black box.

LATE BASIC

The Commodore 64, Apple II, TRS-80, and other microcomputers of the late 1970s and early 1980s featured BASIC in its heyday. Even though Kemeny and Kurtz focused on minicomputer BASIC, it was during this phase

```
{190} 10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

of BASIC's run that the language truly fulfilled many of their goals: ease of use, distribution to millions of users, and availability on a wide variety of platforms. Since that time, the use of BASIC has declined thanks to changing technology, new standards, and a reputation (deserved or not) for encouraging low-quality code among programmers. Modern programming environments are indebted to BASIC in a variety of ways, however.

The most direct lineage continues Microsoft's history of building tools to support the BASIC language. Compilers and development environments supporting BASIC, including QuickBasic and QBasic, shipped with every Microsoft operating system until Windows 2000 finally broke the chain by moving away from an MS-DOS base. In 1991 Microsoft reenvisioned BASIC to produce Visual Basic, a language that was intended to fulfill the ease of use and rapid development capabilities of BASIC under the new paradigm of window-based interfaces. Visual Basic used some syntax similar to BASIC but was designed for use with graphical development tools and did not derive directly from earlier microcomputer BASICs. Visual Basic itself was followed ten years later by Visual Basic .NET, a language that again breaks from its predecessor in fundamental ways but retains the goal of being the easy-to-learn, quick-to-use introductory programming language on the Microsoft platform. As of 2012, Visual Basic is the seventh most popular programming language in the world and Visual Basic .NET is twenty-fourth (TIOBE Software BV 2012).

On the less professional end, Microsoft's most recent BASIC probably has the strongest relationship to **10 PRINT** and to how that program was used, modified, shared, and explored. This version of the language is Microsoft Small BASIC, released in 2008 and available free of charge. This is a Microsoft .NET language that is clearly aimed at students and other beginners. It incorporates turtle graphics concepts from LOGO, inviting play and exploration with graphics and GUI elements as well as text. To accompany this language, there is even a Small BASIC edition of David Ahl's *Basic Computer Games* (Kidware Software 2011).

BASIC has continued to be relevant in particular domains. There are several BASICs, or BASIC-derived languages, created specifically for game development and still in active development and use. These include Blitz BASIC (and successor languages), DarkBASIC, and GLBasic. Those interested in physical computing projects can use a microcontroller, the BASIC Stamp, versions of which have been manufactured by Parallax, Inc. since

1992. This system is powered by a nine-volt battery; hobbyists can program it in a variant language called PBASIC. A less direct descendant is the language used to program calculators from Texas Instruments in the 1990s and 2000s. It has been given the unofficial name of TI-BASIC by its programming community because, as in the heyday of BASIC, it is a relatively simple interpreted language that ships with and controls a stand-alone device.

Other successors have continued to migrate either BASIC's principles or syntax to an ever-widening array of environments. Like Microsoft's Visual Basic, True BASIC updated the BASIC language to support graphical environments. Unlike Microsoft's re-envisioning, however, True BASIC was created by Kemeny and Kurtz themselves and has remained close to both the original syntax of Dartmouth BASIC and the principle of device independence, with compilers available for several operating systems.

A more radical interpretation of BASIC's legacy might include languages that have taken over its role of inviting ordinary users to become programmers and creators. Following the release of graphical web browsers like NCSA Mosaic, Netscape Navigator, and Microsoft Internet Explorer between 1993 and 1995, that role might be assigned to HTML. Though HTML is a markup language used for formatting, not a programming language used for data processing and flow control, it copied BASIC's template of simplicity, similarity to natural language, device independence, and transparency to become many users' first introduction to manipulating code. Browsers have traditionally contained a "view source" command that shows the markup behind the page being displayed, making it as accessible as if it were printed in a magazine. This markup language also was similar to BASIC in that it led users on to more powerful languages like Javascript, Perl, and PHP as those users sought to create more interactivity than static HTML could provide.

BASIC's role as a language that introduced users to programming by necessity in the 1980s is now being fulfilled by languages designed specifically for education, some of which are so abstracted from traditional programming practices that they use entirely different metaphors. Scratch, an environment developed by the MIT Media Lab in 2006 whose creators cite 1980s-era BASIC as a predecessor (Resnick et al. 2009, 62), does not even use text as the basic unit; instead, programs are assembled by dragging and dropping puzzle-piece graphics that fit together to build functional-

A PERSONAL MEMORY OF 10 PRINT

When one of this book's authors, Nick Montfort, first wrote about a similarly functioning program, he presented this variant: `10 PRINT CHR$(109+RND(1)*2); : GOTO 10`. That is the program discussed very briefly in the article "Obfuscated Code" in *Software Studies: A Lexicon*, and is the same version of the program that Montfort presented to Mark Marino's online Critical Code Studies Workshop in 2010, where it sparked the discussion that led to this book.

This program is a different sequence of characters, but it does the same thing as the `10 PRINT` that forms this book's title, for two reasons: first, 205 and 206 are mapped to the same characters as 109 and 110; second, adding a random number between 0 and 2 does the same thing, due to rounding, as adding .5 and then also adding a random number between 0 and 1. The version used in the title of this book is based on (although not identical to) two early print sources for the program, the three-line program in the *Commodore 64 User's Guide* and the one-line version in *RUN* magazine. No print sources from the 1980s have been located that use `RND(1)*2` or that use character codes 109 and 110 rather than 205 and 206.

Why did Montfort initially bring up this "corrupt" version of the program? Simply because he reconstructed `10 PRINT` from memory and looked at a chart of PETSCII character values when he was doing so. Since 109 and 110 are lower numerically and closer to the values for the characters A–Z, he noticed them first on the chart and used those values.

The discussion of this program throughout this book is based on the early print versions. The version of the program that started the discussion, however, came from memory.

ity. Though the appearances and mechanisms are quite different, Scratch uses the same underlying logic and concepts as any other programming language, so that students who use it can apply what they learn to other languages.

Because BASIC was a hit at a unique time in the history of computing—when microcomputers were becoming mainstream but before executable software distribution became widespread—there may never be another language quite like it. The principles behind BASIC remain strong,

though, and continue to make programming languages easier, more transparent, and more freely distributed—all of which continue to encourage new programmers to take the plunge and old programmers to experiment with new ideas.

10 PRINT CHR\$(205.5+RND(1)); : GOTO 10

By: Nick Montfort, Patsy Baudoin, John Bell, Ian Bogost, Jeremy Douglass, Mark C. Marino, Michael Mateas, Casey Reas, Mark Sample, Noah Vawter

Citation:

10 PRINT CHR\$(205.5+RND(1)); : GOTO 10

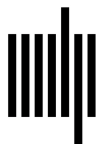
By: Nick Montfort, Patsy Baudoin, John Bell, Ian Bogost, Jeremy Douglass, Mark C. Marino, Michael Mateas, Casey Reas, Mark Sample, Noah Vawter

DOI: 10.7551/mitpress/9040.001.0001

ISBN (electronic): 9780262305501

Publisher: The MIT Press

Published: 2014



The MIT Press

© 2013 Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

MIT Press books may be purchased at special quantity discounts for business or sales promotional use. For information, email special_sales@mitpress.mit.edu or write to Special Sales Department, The MIT Press, 55 Hayward Street, Cambridge, MA 02142.

This book was designed and typeset by Casey Reas using Avenir by Adrian Frutiger, C64 by Style, and TheSansMono by LucasFonts. Printed and bound in the United States of America.

An electronic version of this book is available under a Creative Commons license.

Library of Congress Cataloging-in-Publication Data

10 PRINT CHR\$(205.5+RND(1)); : GOTO 10 / Nick Montfort . . . [et al.].

p. cm.—(Software studies)

Includes bibliographical references and index.

ISBN 978-0-262-01846-3 (hardcover : alk. paper)

1. BASIC (Computer program language)—History. I. Montfort, Nick.

QA76.73.B3A14 2013

005.26'2—dc23

2012015872

10 9 8 7 6 5 4 3 2